5 CACHE POWER BUDGETING

> *The leakage power of a modern last-level cache is larger than the power of a simple core running full out.*
>
> — Mark Horowitz [105]

5.1 Overview

Caches improve performance by reducing the effective memory access latency, but consume significant static and dynamic power. Just as caches cannot be simultaneously large and fast, they also cannot be both large and low power. Smaller caches consume less static power, but can degrade performance and increase dynamic power due to more misses. There is thus a tradeoff between performance and power consumption, which depends on the currently executing set of workloads and data sets. Previous work has shown that workloads often have critical working sets [233], and by reducing the cache size to just hold the working set it is often possible to save power without a significant performance loss [10]. The recent Ivy Bridge microarchitecture powers down a subset of cache ways during periods of low activity [116].

In this chapter we explore power-performance management by dynamically configuring core frequency and resizing the last-level cache. We develop a new governor that targets SLApower, that is, maximizes performance for a power budget (see Figure 3.1 in Chapter 3 for an illustration of the actions that this governor must take). The power budget that we consider is the system power consumed by the baseline configuration that uses a large 32MB last-level cache and runs the cores at 2.132 GHz (see Section 5.2 for details about configurations) to execute a given workload.

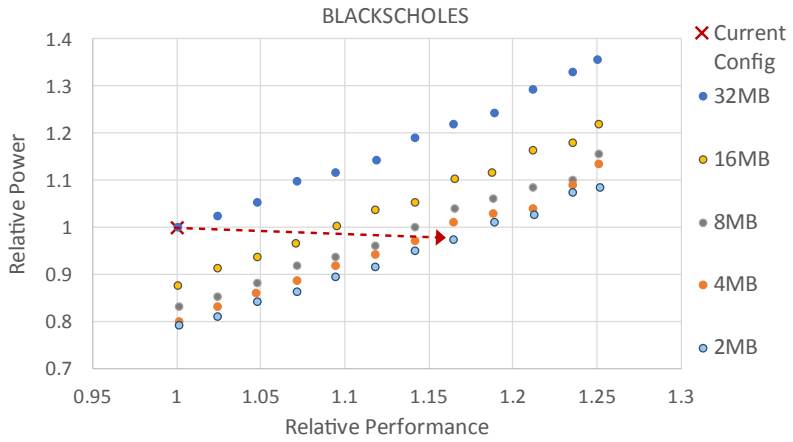


Figure 5.1: Power-performance for blackscholes with DVFS and cache resizing.

Figure 5.1 shows the power-performance state space for one of our workloads, blackscholes, with cache sizes ranging from 2MB to 32MB and core frequencies ranging from 2.132 GHz to 2.665 GHz. The point (1,1), marked with ×, corresponds to the baseline (current) configuration. For a fixed cache size, both performance and power increase with frequency. For this workload, when frequency is fixed and cache size is increased, performance barely changes but power consumption increases. The dashed arrow shows the action needed to be taken by the governor to maximize performance while not exceeding the power of the baseline (current) configuration. For this workload, the cache needs to be reconfigured to 2MB and the frequency needs to be set to 2.482 GHz to get a performance improvement of 16.5% while staying within the power budget. The arrow is slightly dipped instead of being perfectly horizontal due to the unavailability of a valid configuration at that point. Thus, for this workload, the entire power budget cannot be utilized and the desired configuration will save (under-utilize) 2.6% power in addition to improving performance by 16.5%.

Improved performance at the same power translates into energy savings by reducing

RUE and subsequently, CPUE. Depending on the system, cache reconfigurability may also reduce E_{min}. For example, we see in Figure 5.1 that having a 2MB configuration lowers the Pareto frontier compared to the 32MB configuration. This in turn can change EOP (decrease slope of line) and reduce E_{min}. Cache to core power-shifting benefits will be more pronounced on processors running at lower clock frequencies than at higher frequencies. This is because the cache power available to be redistributed is a larger fraction of the system power in such environments whereas core dynamic power dominates system power at higher frequencies.

To intelligently budget power between cores and caches, we investigate using hardware support to drive analytical models of system power and performance. Online estimation enables real-time feedback and adaptation to dynamic changes such as operating system interactions or changing workload mixes. We demonstrate an integrated framework that combines a cache reuse model, performance model, power model and DVFS model to identify optimal power-budgeted configurations.

We extend the state of the art in two ways:

1. We show that careful cache power budgeting—using DVFS and cache reconfiguration—driven by *low-overhead* predictors can improve performance, not just save power. Our new governor exploits this to make the system operate close to Dynamic EO and satisfy SLApower. For our target system, budgeting for system power improves performance by 0.5%–15.2% for 15 of 32 workloads and saves total energy (that includes wall power, not just on-chip and memory power) by 4.4% averaged over all 32 workloads.

2. We develop the first online analytic power and performance models for reconfigurable caches that work for practical replacement policies (i.e., PLRU not just true

LRU), do not use shadow tags, can predict for configurable-set caches, and can configure up to larger caches, not just down to smaller ones.

Earlier works, e.g., Meng et al. [155], have used way counters and shadow tags to determine appropriate cache configurations. However, way counters model fixed-set configurable-associative caches. We discuss in Section 5.5 that such caches can cause large performance degradation with small LLCs in an inclusive hierarchy that can be avoided by using an LLC with large associativity for the same cache size. Thus, we use configurable-set fixed-associative caches in this study. Our reuse distance based cache performance predictor, described in Chapter 4, can be used to model such caches.

The rest of this chapter is organized as follows. Section 5.2 describes the system model and the workloads that we use for our evaluation. Section 5.3 shows some of the opportunities for saving power and energy by reconfiguring caches and the need for intelligent power budgeting. Section 5.4 presents a big-picture view of our power-budgeting approach. Section 5.5 justifies our choice of using configurable-set fixed-associative caches and discusses errors in cache miss rate predictions. Section 5.6 presents the models that we use to predict performance and power. Section 5.7 combines these with DVFS models to predict optimal system configurations.

5.2 Infrastructure

Table 5.1 describes the 8-core CMP we use in this study. We assume that the core frequencies can be increased from the baseline frequency of 2132 MHz in steps of 50 MHz. Section 5.7 describes the scaling assumptions in more detail. Similar to our Haswell server, HS, all cores operate at the same frequency.

Core configuration		4-wide out-of-order, 128-entry window, 32-entry scheduler		
Number of cores		8	**On-chip frequency**	2132–2665 MHz
Technology Generation		32nm	**Temperature**	340K–348K
Functional Units		4 integer, 2 floating-point, 2 mem units		
Branch Prediction		YAGS 4K PHT 2K Exception Table, 2KB BTB, 16-entry RAS		
Disambiguation		NoSQ 1024-entry predictor, 1024-entry double-buffered SSBF		
Fetch		32-entry buffer, Min. 7 cycles fetch-dispatch time		
Inclusive	**L1I Cache**	private 32KB 4-way per core, 2 cycle hit latency, ITRS-HP		
	L1D Cache	private 32KB 4-way per core, 2 cycle hit latency, ITRS-HP		
	L2 Cache	private 256KB 8-way per core, 6 cycle access latency, PLRU, ITRS-LOP		
	L3 Cache	shared, configurable 2–32 MB 32 way, 8 banks, 18 cycle access latency, PLRU, ITRS-LOP, serial		
Coherence protocol		MESI (`Modified`, `Exclusive`, `Shared`, `Invalid`), directory		
On-Chip Interconnect		2D Mesh, 16B bidirectional links		
Main Memory		4GB DDR3-1066, 75ns zero-load off-chip latency, 2 memory controllers, closed page, pre-stdby		

Table 5.1: System configuration.

We assume an 8-banked L3 cache that is dynamically re-configurable, with capacities ranging from 2MB to 32MB and 32-way associativity, for a total of 5 cache configurations. We conservatively assume that the access latency in cycles is constant for all configurations. To evaluate power and performance, we perform full-system simulation using GEMS [149] augmented with a detailed timing and power model. We use CACTI 5.3 [197] to determine the static power and dynamic activation energy per component.

The conventional bit-selection cache index function may not map addresses uniformly across the sets. A number of systems that target commercial workloads use more sophisticated hashing functions [46, 198, 221] to reduce conflict misses. Complex hashed index functions are more common in L2 and L3 caches, where an XOR-based hash function adds a negligible delay to the access latency. We use a simple XOR-based hashing function, discussed in Chapter 4, to distribute lines more uniformly among the L3 cache sets [55, 56]. This usually improves performance over the conventional bit-

Multithreaded		Multiprogrammed	
Workload	**Abbrv.**	**Workload**	**Abbrv.**
blackscholes	blac	astar(4)+bwaves(4)	as-bw
bodytrack	body	astar(4)+gcc(4)	as-gc
fluidanimate	flui	astar_lakes(8)	as-la
freqmine	freq	bwaves(8)	bwaves
swaptions	swap	bzip2(8)	bzip2
ammp	ammp	cactusADM(2)+mcf(2)+milc(2)+bwaves(2)	c-m-m-b
equake	equa	gcc_166(8)	gcc
fma3d	fma3	gcc(1)+omnetpp(1)+mcf(1)+bwaves(1)+ lbm(1)+milc(1)+cactusADM(1)+bzip2(1)	g-o-m-...
gafort	gafo	lbm(8)	lbm
mgrid	mgri	libquantum(8)	libq
swim	swim	libquantum(4)+bzip2(4)	li-bz
wupwise	wupw	mcf(4)+bwaves(4)	mc-bw
apache	apac	mcf(4)+libquantum(4)	mc-li
jbb	jbb	omnetpp(8)	omnetpp
oltp	oltp	omnetpp(4)+lbm(4)	om-lb
zeus	zeus	soplex_pds_50(8)	soplex

Table 5.2: Workloads and their abbreviations. Numbers in parentheses for multiprogrammed workloads indicate the number of copies of the corresponding constituent workload. For example, astar(4)+bwaves(4) means 4 copies of astar and 4 copies of bwaves. Each constituent workload of multiprogrammed workloads is single threaded.

selection index function and also makes cache modeling easier by making the distribution more uniformly random. Due to the self-canceling property of XOR, no extra tag bits need to be stored to retrieve original addresses, if necessary, from the hashed values.

We use two types of workloads—multithreaded and multiprogrammed. Table 5.2 enumerates these workloads along with their abbreviations that we use in the rest of this chapter. Each workload uses a total of 8 threads.

Multithreaded workloads consist of 7 SPEComp [15] benchmarks (ammp, equake, fma3d, gafort, mgrid, swim, wupwise) with "ref" inputs, 5 PARSEC [30] benchmarks (blackscholes, bodytrack, fluidanimate, freqmine, swaptions) with "simlarge" inputs, and 4 Wisconsin

commercial workloads [4] (apache, jbb, oltp, zeus). Each workload runs for a fixed amount of work (e.g. #transactions or loop iterations [6]), corresponding to ~380M – ~570M instructions (average: ~500M) depending on the workload, that is logically partitioned into adjacent training and prediction intervals of roughly equal size. The interval sizes vary according to the available work units in each workload. (See Section 5.4 for a discussion of execution intervals.)

Multiprogrammed workloads consist of combinations of SPEC CPU2006 [101] benchmarks (astar, bwaves, bzip2, cactusADM, gcc, lbm, libquantum, mcf, milc, omnetpp, soplex). Each workload consists of 8 programs, equally divided among the benchmarks in the combination. The training and prediction intervals are ~250M instructions each.

Each simulation run starts from a mid-execution checkpoint that includes cache warmup. Simulating one or more seconds of target execution is infeasible due to high simulation overheads of detailed models.

5.3 Cache Resizing Opportunities

Figure 5.2 shows MPKI (Misses Per Kilo Instruction) with respect to cache size for our workloads in the Prediction Interval. (See Section 5.4 for a discussion of execution intervals.) Our workloads exhibit a range of miss rates from very small (<< 1 MPKI) to quite large (> 40 MPKI). We also observe the following distinctive characteristics among our workloads.

1. **Cache insensitive (low/medium locality)**: those that have a significant part of their working sets that never fit in the cache. They have a high miss rate that changes very little with cache size e.g., equa, fma3, flui, freq, gafo, wupw, libq, lbm.

2. **Cache insensitive (high locality)**: those whose working sets mostly fit in the cache.

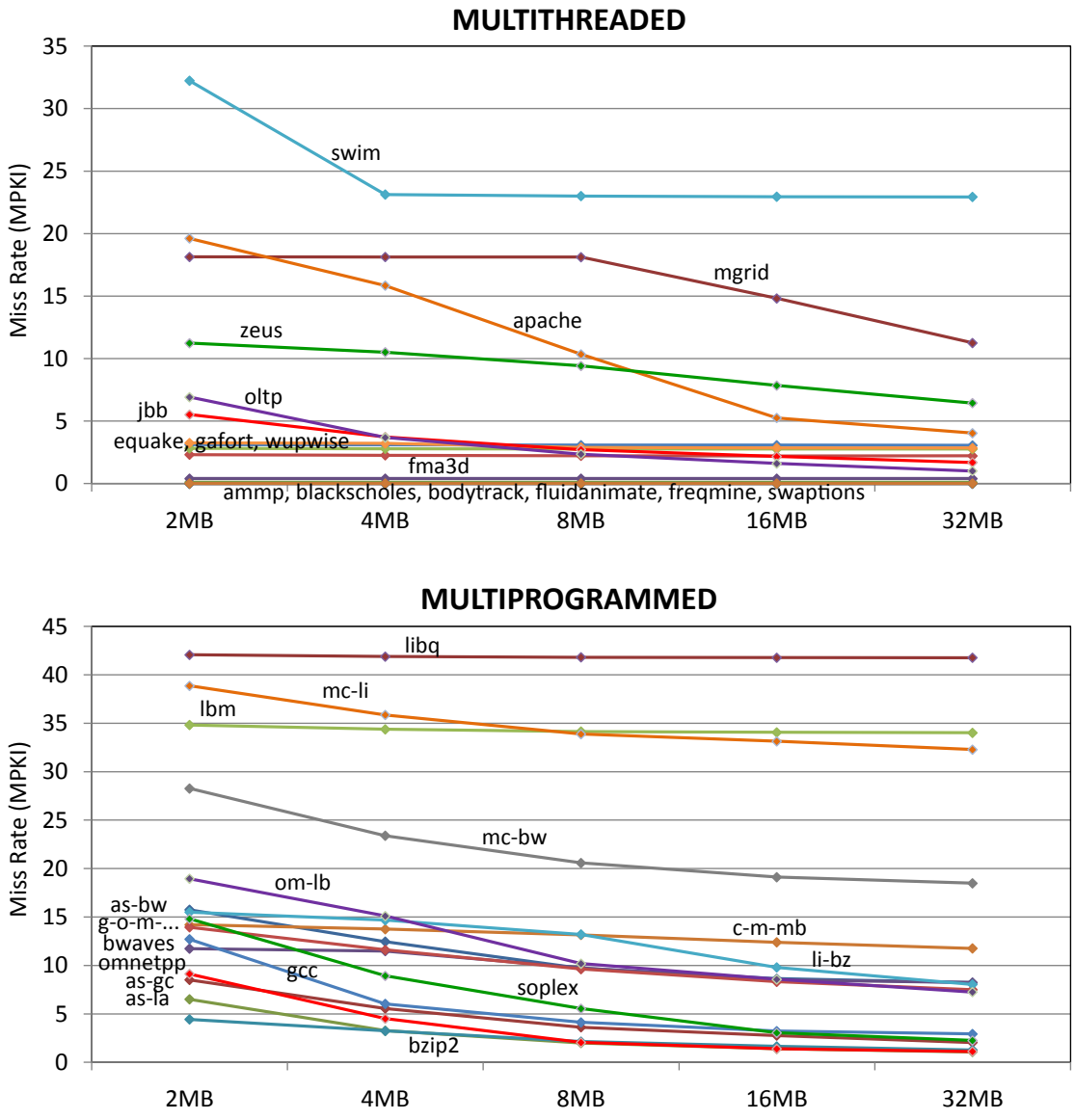


Figure 5.2: MPKI vs cache size (32-way) in the Prediction Interval. (See Section 5.4 for a discussion of execution intervals.)

They have a low miss rate that changes very little with cache size e.g., blac, body, swap, ammp.

3. **Cache sensitive**: those whose working sets fit significantly more in large caches than in smaller ones. Their miss rates change significantly with cache size, e.g., commercial workloads, mgrid, li-bz, mc-bw, mc-li, om-lb, soplex, etc.

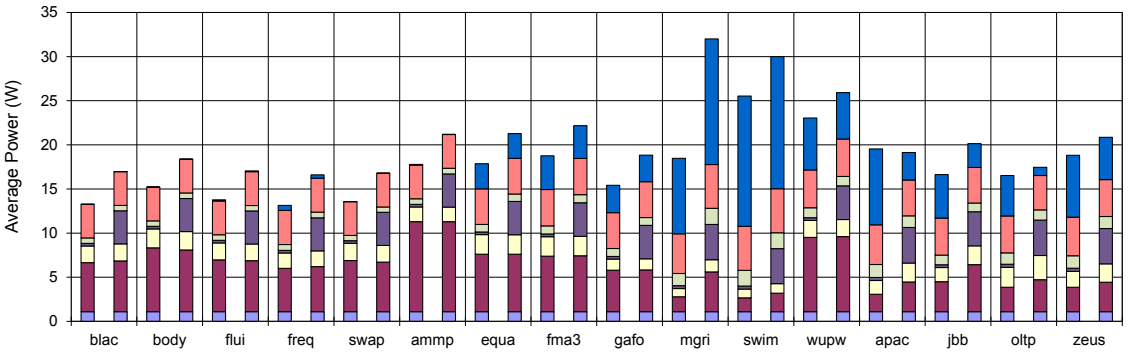
Cache sensitive workloads incur high performance losses with small caches and require larger caches for good performance. So there is less power-saving (and subsequently, power-budgeting) opportunity by using a smaller cache. In contrast, cache insensitive workloads are good candidates for power-budgeting, since their performance largely does not change for different configurations.

Figure 5.3 illustrates the opportunity. It shows the average power breakdowns for our workloads with the smallest LLC configuration (2MB 32-way), the largest LLC configuration (baseline 32MB 32-way), and the performance gain, power saved, and energy saved by executing the workload with the smallest cache instead of with the largest cache. The metrics are inter-related as follows.

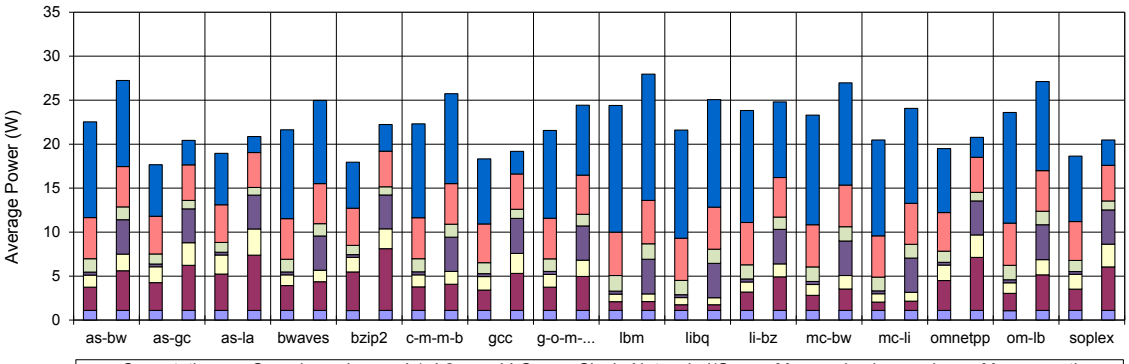
$$\begin{aligned} \text{PerfGain} &= \text{Speedup} - 1 \\ &= \frac{\text{time with 32MB LLC}}{\text{time with 2MB LLC}} - 1 \qquad (5.1) \\ \text{PwrSave} &= 1 - \frac{\text{power with 2MB LLC}}{\text{power with 32MB LLC}} \qquad (5.2) \\ \text{EnrSave} &= 1 - \frac{\text{energy with 2MB LLC}}{\text{energy with 32MB LLC}} \qquad (5.3) \end{aligned}$$

Since Energy = Time $\times$ Power, we have

$$(1 - \text{EnrSave}) = \frac{1 - \text{PwrSave}}{1 + \text{PerfGain}} \qquad (5.4)$$



Workload	blac	body	flui	freq	swap	ammp	equa	fma3	gafo	mgri	swim	wupw	apac	jbb	oltp	zeus
PerfGain	0.1%	-0.2%	-0.5%	-3.4%	-0.1%	0.6%	0.3%	-0.8%	-0.7%	-57.5%	-21.1%	-1.1%	-39.7%	-35.2%	-28.0%	-23.3%
PwrSave	21.8%	17.2%	19.3%	20.8%	19.2%	16.1%	16.0%	15.4%	18.1%	42.3%	14.9%	11.1%	-2.1%	17.5%	5.3%	9.8%
EnrSave	21.8%	17.0%	18.8%	18.1%	19.1%	16.6%	16.3%	14.7%	17.5%	-35.9%	-7.9%	10.1%	-69.4%	-27.4%	-31.4%	-17.6%
EnrSaveW	9.5%	7.6%	7.9%	5.7%	8.1%	8.7%	8.4%	7.2%	7.7%	-73.0%	-15.0%	5.4%	-67.6%	-41.2%	-35.5%	-24.0%



Workload	as-bw	as-gc	as-la	bwaves	bzip2	c-m-m-b	gcc	g-o-m-...	lbm	libq	li-bz	mc-bw	mc-li	omnetpp	om-lb	soplex
PerfGain	-38.5%	-36.8%	-33.1%	-12.6%	-36.1%	-9.4%	-42.0%	-30.6%	-1.3%	0.2%	-40.6%	-26.2%	-6.3%	-42.2%	-49.7%	-49.3%
PwrSave	17.3%	13.5%	9.2%	13.3%	19.2%	13.3%	4.5%	11.8%	12.7%	13.8%	3.9%	13.6%	14.9%	6.1%	13.0%	9.0%
EnrSave	-34.5%	-36.9%	-35.8%	0.9%	-26.4%	4.3%	-64.8%	-27.1%	11.6%	14.0%	-61.9%	-17.1%	9.2%	-62.3%	-72.8%	-79.6%
EnrSaveW	-46.0%	-47.7%	-42.6%	-5.9%	-40.9%	-2.0%	-68.9%	-34.7%	6.5%	7.9%	-64.8%	-24.7%	2.0%	-67.6%	-83.5%	-88.5%

Figure 5.3: Two power stacks are shown for each workload: the left stack for the lowest-power LLC (2MB 2-way) and the right stack for the highest-power LLC (baseline configuration, 32MB 32-way). Power consumed by the 2MB LLC is not conspicuous as it constitutes a small percentage of total power. **PerfGain** shows performance gain (= speedup - 1), **PwrSave** shows system power saved, and **EnrSave** shows system energy saved with the 2MB 32-way cache with respect to the baseline 32MB 32-way cache. **EnrSaveW** accounts for wall power in energy savings calculations. Negative improvements (= losses) are highlighted. Core power includes ROBs, bypass networks, functional units, register files, TLBs, branch predictors, fetch & decode logic. We assume aggressive clock-gating. See Section 5.2 for details on system model.

While PwrSave and EnrSave only account for socket power and memory power, EnrSaveW uses wall power estimates into calculating energy savings. We use a quadratic function, discussed in Section 3.8 of Chapter 3, to estimate wall power from system power reported by the simulator.

As discussed above, cache insensitive workloads show negligible performance impact but significant power (and energy) savings. Cache sensitive workloads, on the other hand, show significant performance and energy losses, e.g., apache suffers a 39.7% performance loss and a 67.6% energy loss (EnrSaveW), soplex suffers a 49.3% performance loss and a 88.5% energy loss (EnrSaveW). Further, using a smaller cache does not always save system power—apache burns 2.1% more power. Overall, 19 of the 32 workloads waste energy if the cache is configured to the smallest size. The geometric mean of energy waste is 8.3% for multithreaded workloads and 33.9% for multiprogrammed workloads.

Thus, power budgeting must be done carefully, as a poor choice of cache configuration can drastically degrade performance and waste energy. Selecting a cache configuration that optimizes power-efficient performance is challenging since the optimal point varies for different workloads, and even for different phases within a given workload. Exhaustive evaluation is impractical due to the large number of feasible system configurations.

In this work, we use analytical models to estimate the power and performance of different configurations, allowing rapid prediction of the optimal system configuration. This requires four predictors: cache miss rate predictor, performance impact predictor, power impact predictor, optimal DVFS scaling predictor. The miss rate predictions are combined with simple performance and power models (Section 5.6) and an on-chip DVFS model (Section 5.7) to identify power-budgeted configurations that will maximize performance.

5.4 Operations overview

This work focuses on *improving* performance by shifting power from the last-level cache to the cores. For cores, we use conventional on-chip DVFS techniques, similar to those used in Intel's Turbo Boost [111], to run cores at higher voltage and frequency. For caches, we use power-gating [162] to eliminate all power for disabled cache regions.

While power-gating can enable a higher power margin for utilization, it results in dirty data being written back to memory when down-sizing the cache and non-trivial warmup time after up-sizing the cache. Assuming a sustained memory bandwidth of 8.53 GBPS (half of maximum bandwidth in the baseline) a worst-case scenario with 32MB of dirty data needs ~3.8 msec writeback time. To keep performance and energy overheads small ($< 1\%$) the reconfiguration interval should be at least 380 msec. *This automatically precludes reacting to high frequency events such as context switches that may occur between 300 to 5000 times per second* [64]. In contrast, drowsy mode [77] can enable small reconfiguration intervals with low overhead, but its need to maintain a minimum data retention voltage (DRV) limits the available power margin.

Reconfiguration being *costly* and *infrequent*, trial-and-error search for the optimal configuration at runtime is not appealing. Stepwise adaptation [163] may progress through multiple intermediate states. In contrast, this work uses online power-performance predictors that enable *one-shot* reconfiguration. They have the following strengths:

- Ability to predict performance for caches larger or smaller than the currently active configuration so that the optimal configuration can be reached in one step.
- Ability to predict the effect of changes in the number of sets of the cache.
- Ability to work with implementable cache replacement policies, such as PLRU. Prior work [155] has assumed LRU replacement, which is impractical to implement

1. Concurrently,
   a) Specialized hardware (Section 4.6.1 of Chapter 4) tracks reuse distance distribution for L3 accesses.
   b) Simple hardware performance counters track activity factors of cores and caches.
   c) Simple hardware performance counters track correlation between miss rate and CPI.
2. Periodically (e.g., once per second), invoke a software routine (ISR) to determine optimal cache configuration:
   a) Predict miss rates (Section 5.5) using information from 1(a).
   b) Predict performance and power (Section 5.6) using information from 1(b), 1(c), 2(a).
   c) Predict DVFS scaling and possible gain (Section 5.7) using information from 1(c), 2(a), 2(b).
3. Reconfigure system with the best predicted configuration if predicted gain is $> 2\%$. Write back dirty cache blocks as needed.
4. System continues to monitor predicted and actual performance and power metrics to detect and adjust in case of incorrect predictions (e.g., due to phase change effects).

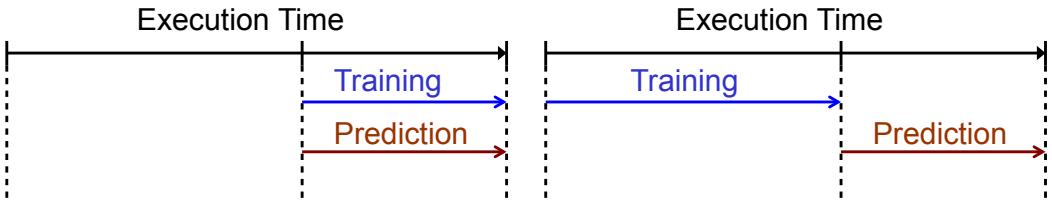
Figure 5.4: Operations Overview.

for highly-associative caches.

Figure 5.4 shows an operational overview of our proposed system. Execution time is logically partitioned into intervals. Like most predictors, our work uses past execution behavior to predict behavior in subsequent intervals. Simple hardware mechanisms are used to observe execution characteristics that are then used by prediction software (ISR) to determine the optimal system configuration. We refer to the interval used to train the predictors as the *Training Interval* and the interval where the results of prediction are applied as the *Prediction Interval*.

Intervals should be long enough so that predictor and reconfiguration overheads are small (< 1%). Predictor compute time depends on the number of target cache configurations evaluated. This has two components: miss-rate prediction (< 0.5 msec, using Section 4.5.5, Chapter 4, results for 25 configurations) and DVFS scaling computation (< 0.3 msec, see Section 5.7.4). Together with cache reconfiguration overhead (< 3.8 msec), the total time overhead < 3.8 + 0.5 + 0.3) = 4.6 msec. A small amount of additional time is needed to read various performance counters. So, we recommend an interval length > 500 msec. Longer intervals corresponding to 1 or more seconds of execution may be needed to get sufficient samples. Our work targets optimizing system operations for the average execution profile over long-term intervals.

There are two sources of error in the predicted values: *model error* and *phase error*. Model error results from simplifying assumptions (e.g., independence and identical distribution) that may not strictly hold in practice. Phase error also includes changes in workload behavior as it moves though different phases of execution [195]. This distinction helps identify which errors could be reduced by further refining the model and which errors are orthogonal to it. When phase changes occur the behavior from a previous interval is not a good predictor for the next interval. The predictors may be improved using previously proposed online phase detection mechanisms [161].



(a) **Model Error Prediction** (b) **Phase Error Prediction**

Figure 5.5: Training and Prediction intervals.

Model Error predictions use the same interval for training and prediction and hence

incur model error alone. Such predictions offer insight into model accuracy, but are useless for reconfiguration purposes. Phase Error predictions use a preceding training interval to predict the behavior of a subsequent prediction interval and includes both model and phase errors. These predictions are used to reconfigure the system. Figure 5.5 illustrates training and prediction intervals. In the rest of this chapter, we will use the term Model Error and Model Error prediction interchangeably and likewise with Phase Error and Phase Error prediction.

For both Model Error and Phase Error predictions, we study *absolute error* and *relative error*. The absolute error is (predicted value - actual value). Relative error is (absolute error/actual value). The actual and predicted values can be inferred using the calculation: actual value = (absolute error/relative error).

5.5 Cache miss rate prediction

Predicting the cache miss rate for all possible cache configurations is critical to our power-budgeting approach. We use our reuse distance based cache miss ratio predictor, described in Chapter 4, for this purpose. *This section elaborates step 2(a) of Figure 5.4.*

Since our simulation runs are not very long, we do not perform address sampling to estimate the reuse distributions. Instead, we assume that the full reuse distributions are available for use as inputs to our miss rate predictors. A practical deployment would need to use sampling, as described in Chapter 4, over longer runs. With a sufficient number of samples, the inferred reuse distributions should be close to the actual distributions.

Figure 5.6 shows reuse distance distributions for our workloads in the Prediction Interval. The dashed vertical lines with size annotations indicate fully-associative LRU cache sizes that would be necessary if all accesses with reuse distances less than or equal to that point must hit. Cache insensitive workloads, such as blac and equa, have “flat”

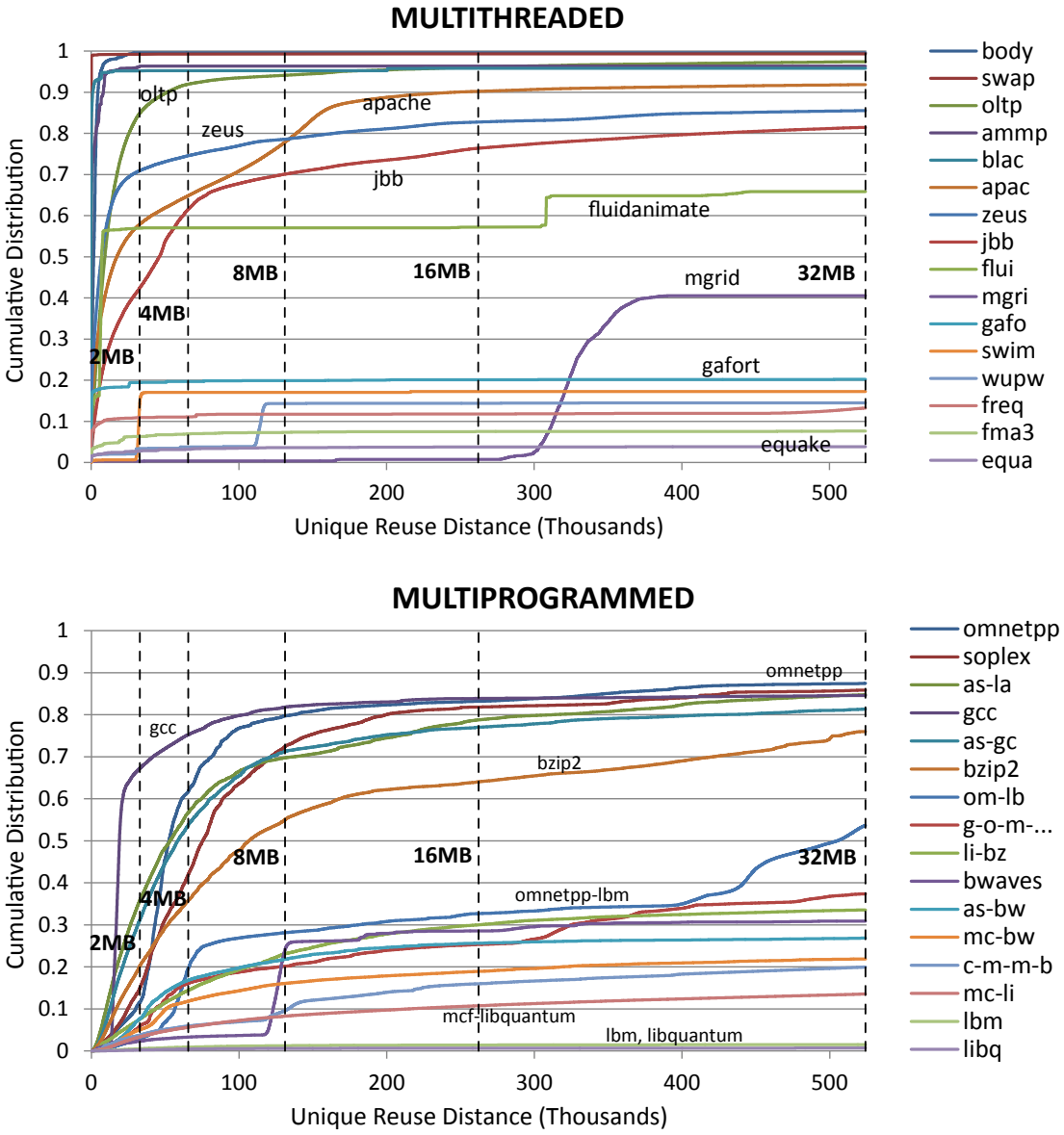


Figure 5.6: Reuse distributions (cumulative) in the Prediction Interval. Entries in the legend are ordered according to the intercept of the distributions on the right vertical axis. Additionally, a few distributions are labeled with the workload name to enhance readability. A 32MB cache has 512K lines.

reuse distributions over the range of cache sizes that we consider whereas cache sensitive workloads show a significant slope.

Good miss rate predictions should not have high absolute errors. That is, the difference between the predicted and actual MPKI should be small. Since cache miss rates significantly affect overall performance, large absolute errors in MPKI prediction will also translate into large errors in performance prediction that can lead to poor configuration selections. Note that relative errors can be large for workloads with small miss rates but have little effect on performance if the absolute errors are small.

For this study, we consider cache reconfiguration in the number of sets, but not in the number of ways. Our caches always have 32-way associativity for all cache sizes. This is because for small caches, smaller associativities usually cause more conflict misses than with larger associativities. Since our cache hierarchy is inclusive, evictions at the LLC can use evictions at L2 which can then result in increased demand accesses from L2 to L3 that subsequently miss in L3. The effect of this is two-fold:

1. 2MB 2-way LLCs can witness significantly more accesses and misses than 2MB 32-way caches while saving negligible power due to reduced associativity, and

2. miss rate predictions using the characteristics of the address stream with the 32MB 32-way LLC are no longer accurate since the nature of the access stream to the LLC is altered.

To elaborate on the above points, we momentarily assume that our LLC is configurable in both the number of sets and ways. Figure 5.7 shows the number of accesses (misses from L2) to a 2MB 2-way LLC and to a 2MB 32-way LLC, normalized to the number of accesses to the LLC in the baseline configuration (32MB 32-way). Figure 5.8 shows the corresponding MPKI values. Ideally, there should be no differences in the number

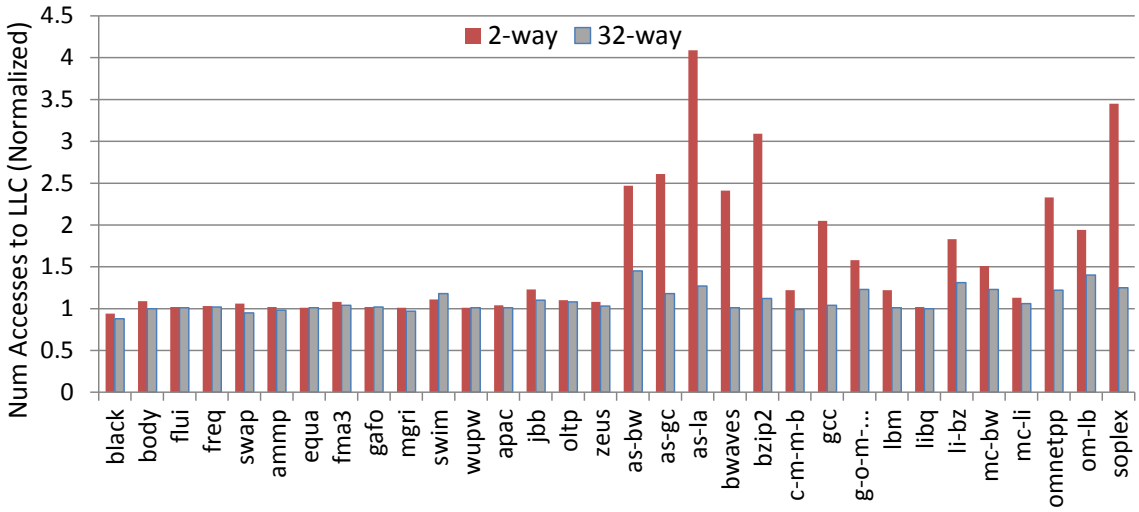


Figure 5.7: Number of accesses for a 2MB LLC with small and large associativities, normalized to the number of accesses to the baseline 32MB 32-way LLC.

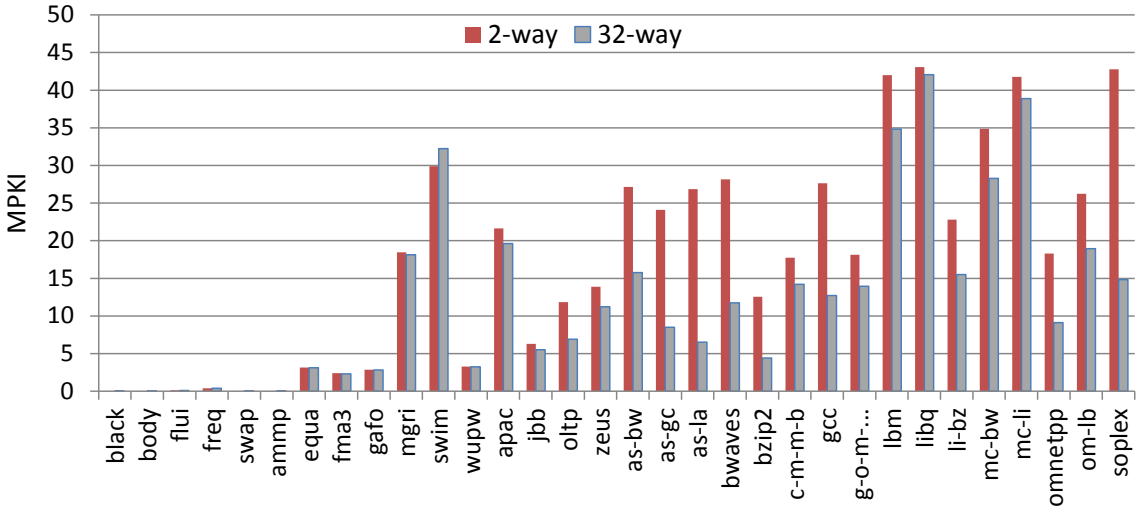


Figure 5.8: MPKI for a 2MB LLC with small and large associativities.

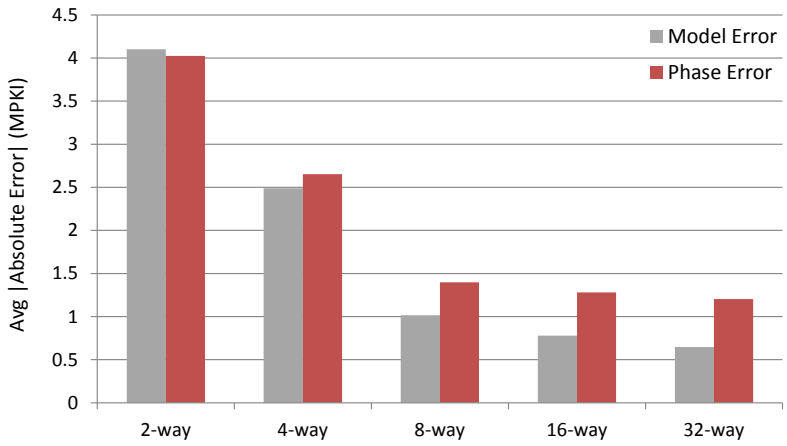


Figure 5.9: Averages of Absolute Error for miss ratio estimation.

of accesses to the LLC. However, due to a strictly inclusive cache hierarchy and more conflicts, the 2-way cache witnesses more accesses than the 32-way cache. The effect is more pronounced for the multiprogrammed workloads than for the multithreaded workloads. Large changes in the access stream contribute to large changes in MPKI values in addition to that due to changed cache organization.

Figure 5.9 shows the average of absolute errors in MPKI prediction for both Model Error and Phase Error over all workloads for different associativities and all cache sizes (2MB, 4MB, 8MB, 16MB, 32MB) for each associativity. We observe that for both Model Error and Phase Error, predictions for 32-way caches are significantly more accurate than for 2-way caches. For Model Error, the average error increases from 0.65 MPKI to 4.1 MPKI—a 6.3× change. For Phase Error, the average error increases from 1.2 MPKI to 4 MPKI—a 3.3× change.

Figures 5.10 and 5.11 show prediction error distributions for Model Error and Phase Error respectively for different associativities over all cache sizes that we consider. The predictions become more accurate at higher associativities. This is seen in the Absolute

Error vs. Relative Error scatter plots where the data points become more concentrated around the origin as associativity increases. It is also seen in the cumulative distribution plots where the worst case error decreases and the curves reach 100% more quickly as associativity increases.

For the rest of this chapter we only consider 32-way caches. Figures 5.10 and 5.11 show that the absolute error for Model Error is < 7 MPKI with 90% of errors within 2.4 MPKI. For Phase Error, the worst-case error is ~10 MPKI with 90% of errors within 3.74 MPKI. The Phase Error charts also show a few points with high relative error (swap) but this happens with at small MPKI values and result in small absolute errors, so the performance impacts of mispredictions are small.

5.6 Performance and Power prediction models

Predicting miss rates for target cache configurations is necessary, but not sufficient for making power-budgeting decisions; it is also necessary to predict performance and power impact. *This section elaborates step 2(b) of Figure 5.4.*

To predict performance given an L3 miss rate r for an instruction interval, let $\hat{CE}(r)$ denote the estimated number of cycles to complete that interval. $\hat{CE}(r) = \hat{CPI}(r) \times$ number of instructions (aggregate, over all cores). We approximate $\hat{CPI}(r)$ by measuring the actual L3 misses, cycles and instructions committed during the training interval using hardware performance counters and then using least-squares regression to fit this to a simple linear model so that $\hat{CPI}(r) = g + r \times h$ for estimated constants g and h.

For a set of n tuples of the form (x_i, y_i), $i = 1..n$, the constants can be calculated with

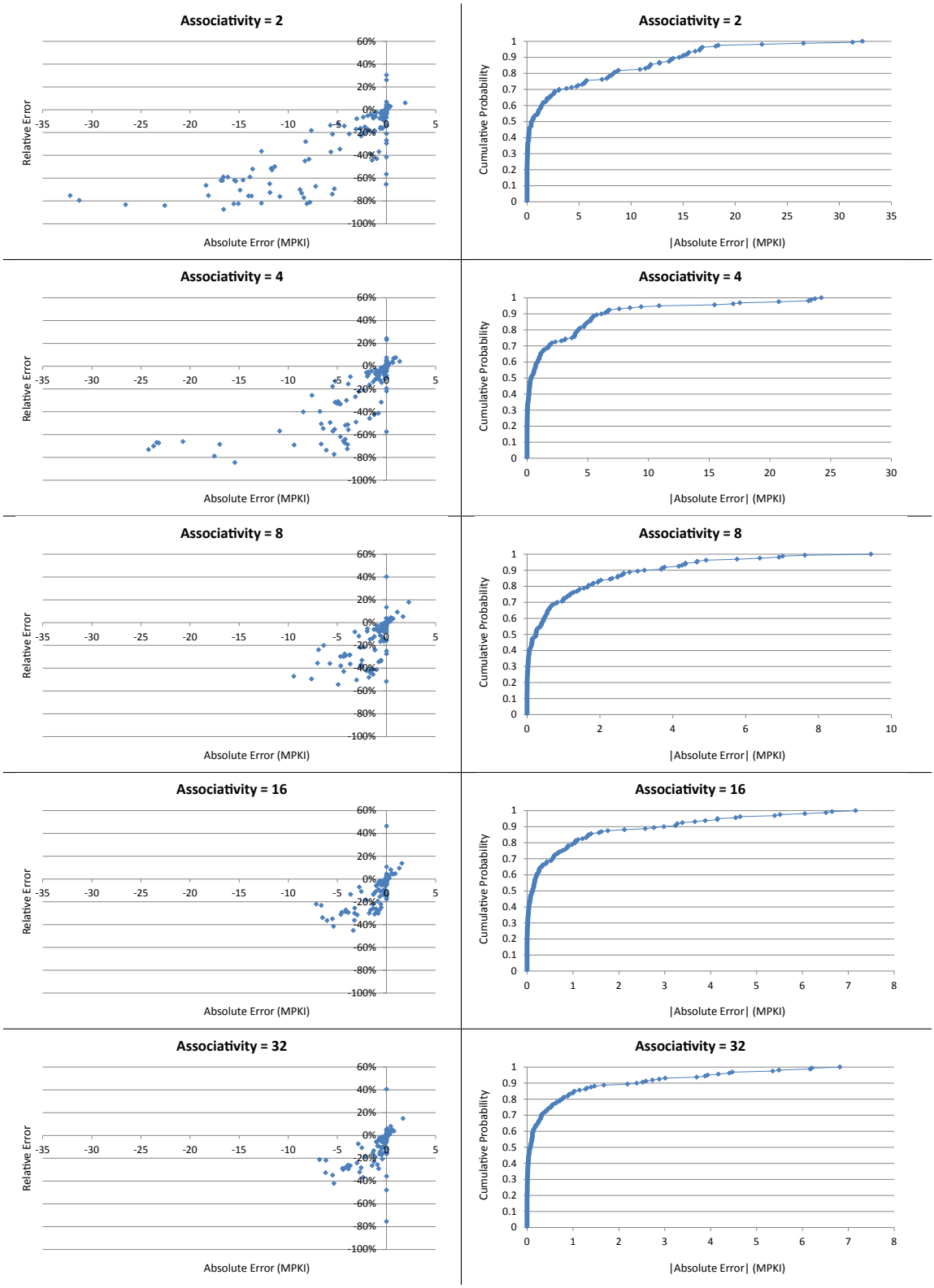


Figure 5.10: Model Error for miss ratio estimation.

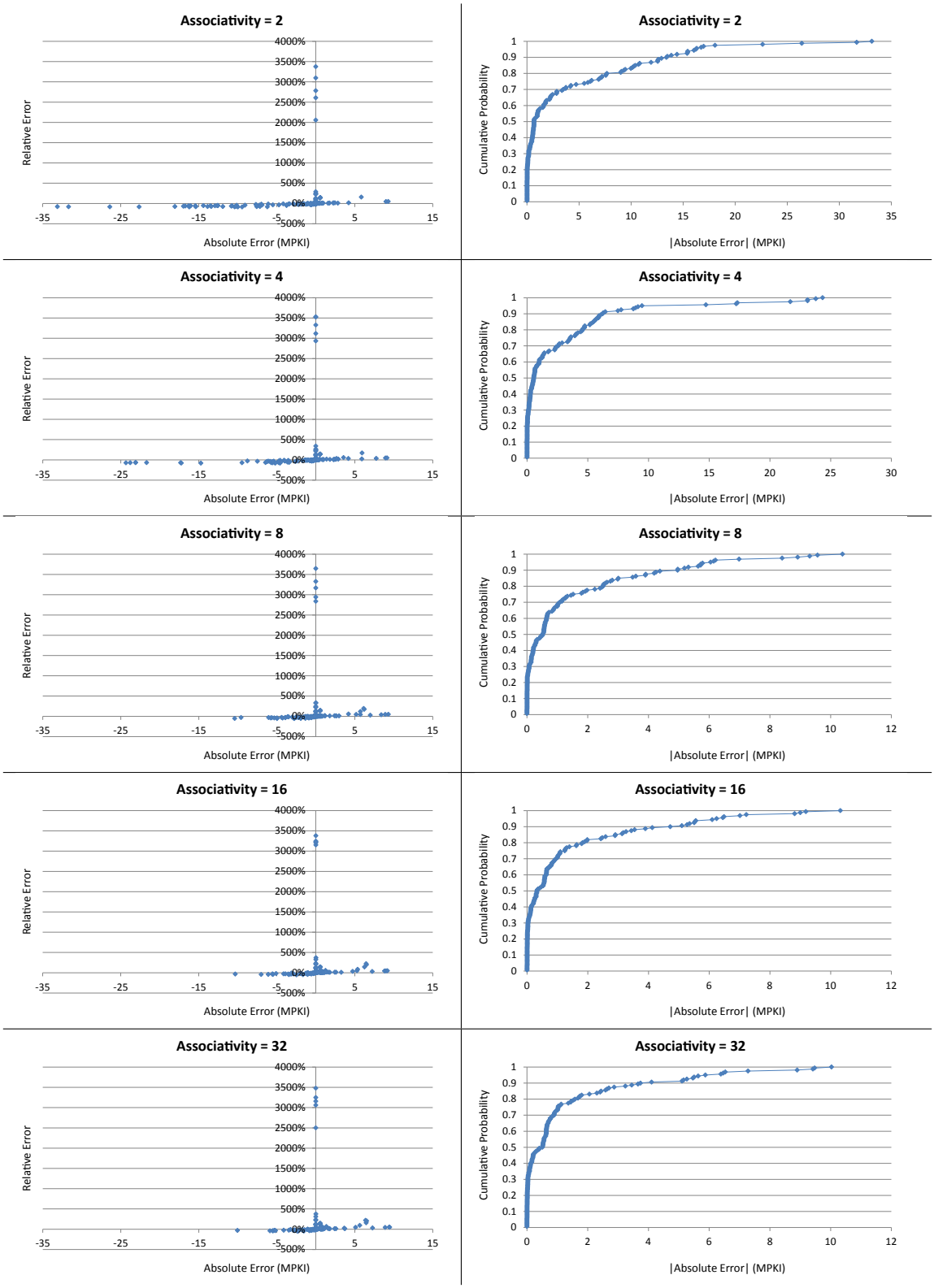


Figure 5.11: Phase Error for miss ratio estimation.

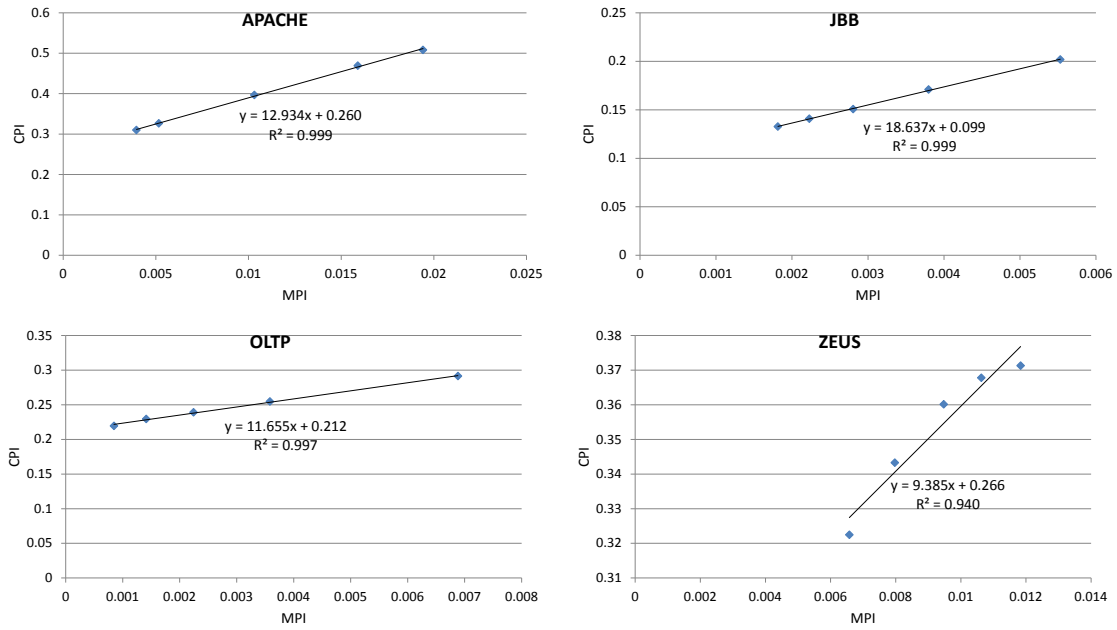


Figure 5.12: CPI regression for commercial workloads in the Training Interval

the following equations.

$$h = \frac{\frac{\sum_{i=1}^{n} x_i y_i}{n} - \left(\frac{\sum_{i=1}^{n} x_i}{n}\right)\left(\frac{\sum_{i=1}^{n} y_i}{n}\right)}{\frac{\sum_{i=1}^{n} x_i^2}{n} - \left(\frac{\sum_{i=1}^{n} x_i}{n}\right)^2} \tag{5.5}$$

$$g = \frac{\sum_{i=1}^{n} y_i}{n} - h\left(\frac{\sum_{i=1}^{n} x_i}{n}\right) \tag{5.6}$$

The above estimation is done online. At the end of every 20,000 cycles, two metrics are computed: (x_i = Δ LLC misses/Δ instructions committed) and (y_i = Δ cycles/Δ instructions committed). The individual (x_i, y_i) tuples are not stored. Instead, cumulative metrics (product, square, sum) with earlier values are computed. 4 registers and 1 counter (for tracking total number of tuples) are maintained. At the end of the training interval, the slope (h) and offset (g) of the best-fit line are computed.

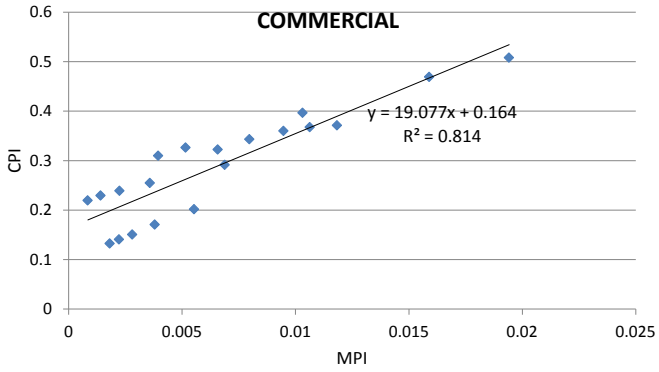


Figure 5.13: Combined CPI regression for commercial workloads in the Training Interval

The linear approximation does not always succeed. In some cases due to variations in program behavior, low MPKI, or lack of variability in the tuples, the slope may be computed as negative. This is unrealistic as we expect execution time to increase with MPKI due to the long off-chip miss latency. In such cases, the computed slope is discarded and instead a predetermined value is chosen. We chose this to be the value estimated for the commercial workloads over the Training Interval using oracle information. We get this oracle information by simulating the workloads for all possible cache sizes in the Training Interval and measuring the MPI and CPI values. Figure 5.12 shows the individually fitted lines as well as R^2 values for each commercial workload. The R^2 values are close to 1 indicating good fits. Figure 5.13 shows the fitted line over data points from all four commercial workloads. The fit is less good (lower R^2 value), but we use this as a representative for average program behavior. Also, we predict CPI to be the same if predicted MPKI is within 0.01 or 1% of the current configuration.

Figures 5.14 and 5.15 show Model Error and Phase Error in CPI estimation over all workloads using the online estimator. The average of absolute values of relative errors for Model Error is 4.7% whereas for Phase Error it is 9.9%. For Model Error, 90% of

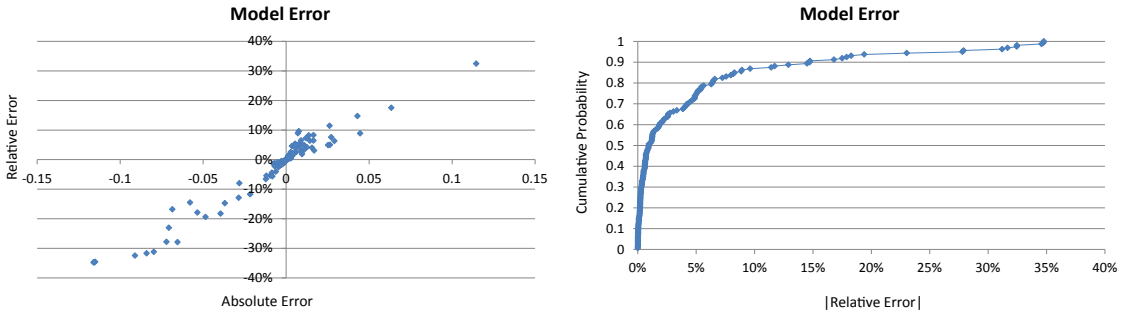


Figure 5.14: Model Error for CPI estimation.

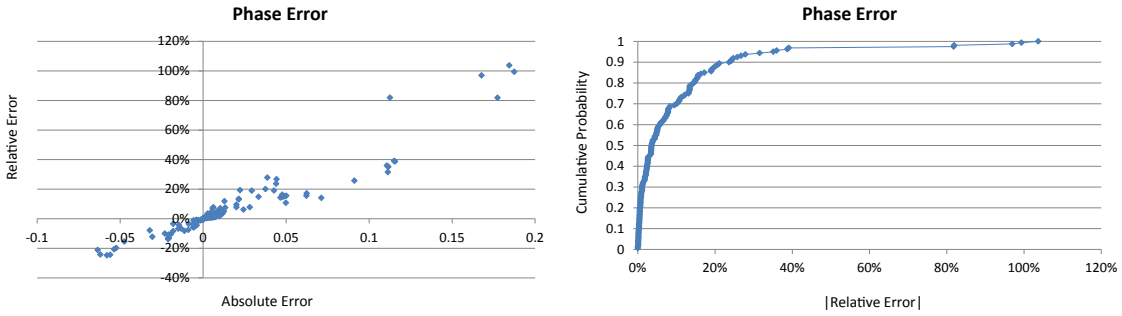


Figure 5.15: Phase Error for CPI estimation.

predictions have within 14.7% relative error whereas for Phase Error it is 23.6%.

For power predictions, we separately predict static power and dynamic power. We use CACTI to estimate static power and dynamic energy per activation per component. For the L3 cache, the number of tag and data activations for accesses, misses, replacements and coherence activities is tracked. To predict activations, the model makes some simplifying assumptions, e.g., L3 accesses, coherence activities, and the percentage of L3 misses that cause additional writebacks to memory is the same as that observed in the current configuration. These assumptions are not strictly true but work reasonably well.

Figures 5.16 and 5.17 show Model Error and Phase Error in system power estimation over all workloads using the online estimator. The average of absolute values of relative

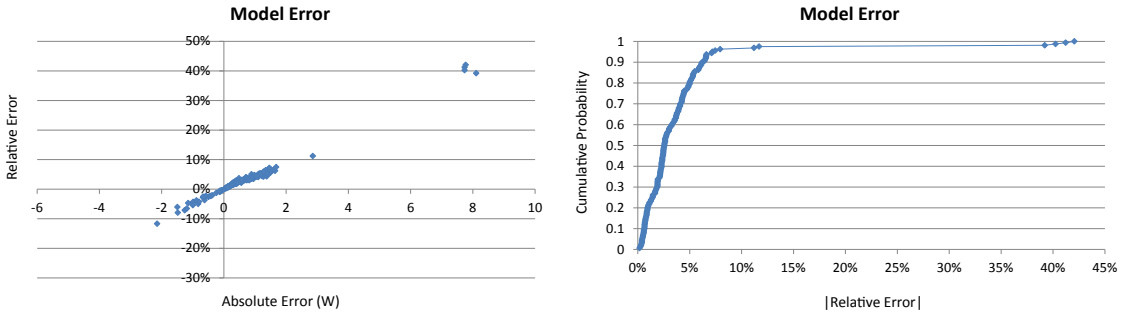


Figure 5.16: Model Error for system power estimation.

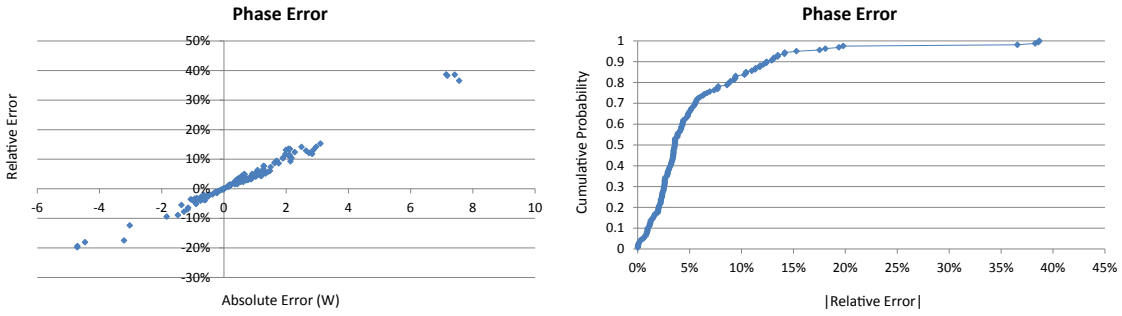


Figure 5.17: Phase Error for system power estimation.

errors for Model Error is 4% whereas for Phase Error it is 5.8%. For Model Error, 90% of predictions have within 6.2% relative error whereas for Phase Error it is 12.4%. Since static power is known, the errors are due to dynamic power estimations. This has two components: activation count estimation and performance estimation. Improving either will reduce errors.

5.7 Model-driven Power Budgeting

The analytical models of the preceding sections determine the power and performance of different cache configurations for a given workload. Here we describe how to estimate whether or not it is better to reconfigure the cache to a smaller configuration that uses less

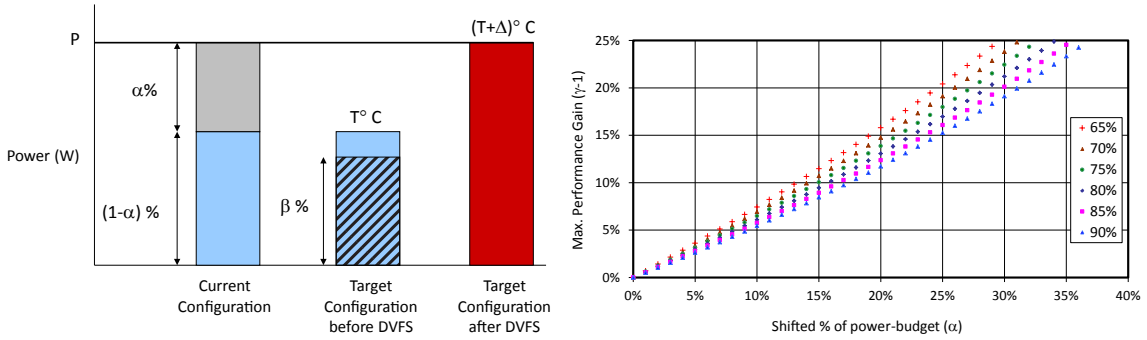


Figure 5.18: Max. performance gains with basic model (Section 5.7.1). Each series corresponds to a different value of β, from $\beta = 65\%$ to $\beta = 90\%$. $D_{old} = \beta(1-\alpha)P$.

power, leaving more power available to run the core at a higher voltage and frequency. *This section elaborates step 2(c) of Figure 5.4.*

We develop our power-budgeting model in three steps, each step adding some more complexity to the previous one. First, Subsection 5.7.1 presents a basic model for calculating maximum performance gains in a power-budgeting environment where power gating and on-chip DVFS are used for shifting power. Next, Subsection 5.7.2 applies it to power budgeting for on-chip resources and includes the effects of temperature rise with frequency scaling on static power estimations. Finally, Subsection 5.7.3 includes main memory power in the power budget. We use only on-chip DVFS. Memory voltage and frequency are not scaled.

5.7.1 Basic model

Let P be the power-budget (assumed to be fully utilized by the current configuration) and let α denote the fraction of this budget that can re-budgeted for better performance. The power margin, αP, can include both static *and* dynamic power of the current configuration. Ideally, DVFS transforms all of αP into additional dynamic power in

the target configuration. However, there are leakage losses due to temperature and voltage rise and performance losses due to non-scaling of memory latency. Figure 5.18 shows a schematic of power shifted from the old (current) configuration to the new (target) configuration. After DVFS, the dynamic power of the target configuration, $D_{new} \leqslant D_{old} + \alpha P$. Since dynamic power is proportional to $V^2 f$, we have

$$\frac{D_{new}}{D_{old}} = \left(\frac{V_{new}}{V_{old}}\right)^2 \left(\frac{f_{new}}{f_{old}}\right) \leqslant \left(1 + \frac{\alpha P}{D_{old}}\right) \tag{5.7}$$

Let $\gamma = \frac{f_{new}}{f_{old}}$. So, *Speedup* $\leqslant \gamma$. Using published data [99] for voltage-frequency pairs for the Pentium M, we assume that $(V_{new} - V_{old}) \propto (f_{new} - f_{old})$ and that every 200MHz change in frequency is accompanied by a 50mV change in voltage. Thus, $V_{new} = V_{old} + 50mV\left(\frac{f_{new} - f_{old}}{200MHz}\right)$. We also assume a base operating voltage of 0.9V and a base operating frequency of 2132MHz. Substituting values in Equation 5.7,

$$\left(\frac{V_{new}}{V_{old}}\right) = (1 + 0.5922(\gamma - 1)) \tag{5.8}$$

$$(1 + 0.5922(\gamma - 1))^2 \gamma \leqslant \left(1 + \frac{\alpha P}{D_{old}}\right) \tag{5.9}$$

Figure 5.18 shows maximum performance gains with $\gamma \leqslant 1.25$. The gains increase with both α and β.

5.7.2 On-chip power-budgeting model

We will now extend the basic model by including the effects of temperature rise with DVFS and will apply it to on-chip power budgeting. For on-chip power-budget P, current operating voltage V, frequency f, temperature T and target cache configuration $\bar{C} = (capacity, associativity)$, let $\hat{P}_{st}(\bar{C}, T)$ and $\hat{P}_{dyn}(\bar{C}, V, f)$ denote the estimated on-chip static and dynamic power respectively. The dynamic power before DVFS,

$D_{old} = \hat{P}_{dyn}(\bar{C}, V, f)$. The power margin, $\alpha P = P - \hat{P}_{st}(\bar{C}, T) - \hat{P}_{dyn}(\bar{C}, V, f)$ can be utilized by increasing the operating voltage and frequency to V_{new} and f_{new} respectively. However, scaling is accompanied by an increase in static power that reduces the available power margin due to temperature and voltage increase.

Since chip power-budget is fixed, ideally, overall chip temperature cannot rise (Stefan-Boltzmann law). However, since processor chips are not ideal black-bodies, the higher dynamic power of the cores can cause the operating temperature to rise locally. This in turn increases the static power dissipation by $\Delta\hat{P}_{st}(T) = \hat{P}_{st}(\bar{C}, T_{new}) - \hat{P}_{st}(\bar{C}, T)$. $\Delta\hat{P}_{st}(T)$ depends on γ. For our implementation we assumed a maximum temperature rise of 8 °C corresponding to a maximum scaling of 533MHz (2132MHz to 2665MHz) with 3 °C contributed by every 200MHz [99]. We assumed a continuous scaling domain with temperature rise proportional to scaling. Thus, $\Delta\hat{P}_{st}(T) = (\frac{\gamma-1}{1.25-1}) \times (\hat{P}_{st}(\bar{C}, T+8) - \hat{P}_{st}(\bar{C}, T))$. By default, CACTI allows modeling temperature effects in steps of 10K. We used linear interpolation to obtain static power at other points. So,

$$\Delta\hat{P}_{st}(T) = \left(\frac{\gamma-1}{0.25}\right) \times \frac{8}{10} \times (\hat{P}_{st}(\bar{C}, T+10) - \hat{P}_{st}(\bar{C}, T)) \tag{5.10}$$

The higher operating voltage increases static power dissipation. Assuming a linear model [38], the increase is $\Delta\hat{P}_{st}(V) = \left(\frac{V_{new}}{V_{old}} - 1\right) \times \hat{P}_{st}(\bar{C}, T))$. Combining with Equation 5.8, we have

$$\Delta\hat{P}_{st}(V) = 0.5922 \times (\gamma - 1) \times \hat{P}_{st}(\bar{C}, T) \tag{5.11}$$

Both $\Delta\hat{P}_{st}(T)$ and $\Delta\hat{P}_{st}(V)$ reduce αP. Plugging values into Equation 5.9, we get

$$(1 + 0.5922(\gamma - 1))^2\gamma \leqslant \left(\frac{P - \hat{P}_{st}(\bar{C}, T) - \Delta\hat{P}_{st}(T) - \Delta\hat{P}_{st}(V)}{\hat{P}_{dyn}(\bar{C}, V, f)}\right) \tag{5.12}$$

Since *memory voltage and frequency are not scaled*, the number of cycles to execute the same task in the scaled configuration is increased due to scaling of memory latency (in terms of processor cycles). This reduces the increase in on-chip dynamic power that Equation 5.12 assumes. We use this fact to improve Equation 5.12. With a linear performance predictor model, $\hat{CPI}(r) = g + r \times h$. After scaling, $\hat{CPI}(r)_{scaled} = g + \gamma \times r \times h$. So,

$$(1+0.5922(\gamma-1))^2\gamma \leqslant \left(\frac{g+\gamma\times r\times h}{g+r\times h}\right) \times \left(\frac{P-\hat{P}_{st}(\bar{C},T)-\Delta\hat{P}_{st}(T)-\Delta\hat{P}_{st}(V)}{\hat{P}_{dyn}(\bar{C},V,f)}\right) \quad (5.13)$$

5.7.3 System power-budgeting model

We will now include memory power in addition to on-chip power in the power budget. We start from Equation 5.13, noting that since memory voltage and frequency are not scaled, $\hat{P}_{dyn}(\bar{C},V,f)$ *still refers to on-chip dynamic power*. However, increase in memory power can reduce the available power budget. Let P', $\hat{P}_{mbp}(\bar{C})$ and $\hat{P}_{map}(\bar{C},\gamma f)$ denote the available system power-budget, estimated memory background power and estimated memory active power. Equation 5.13 can now be reformulated as

$$\begin{aligned}(1+0.5922(\gamma-1))^2\gamma \leqslant \left(\frac{g+\gamma\times r\times h}{g+r\times h}\right) \times \\ \left(\frac{P'-\hat{P}_{st}(\bar{C},T)-\Delta\hat{P}_{st}(T)-\Delta\hat{P}_{st}(V)-\hat{P}_{mbp}(\bar{C})-\hat{P}_{map}(\bar{C},\gamma f)}{\hat{P}_{dyn}(\bar{C},V,f)}\right)\end{aligned} \quad (5.14)$$

For I instructions, the expected execution time with the target configuration is $I \times \left(\frac{g+\gamma\times r\times h}{\gamma}\right)$ with frequency scaling γ and $I \times (g + r \times h)$ without frequency scaling. So,

$$\hat{P}_{map}(\bar{C},\gamma f) = \gamma \times \left(\frac{g+r\times h}{g+\gamma\times r\times h}\right) \times \hat{P}_{map}(\bar{C},f) \quad (5.15)$$

Equation 5.15 is plugged into Equation 5.14 for the final equation.

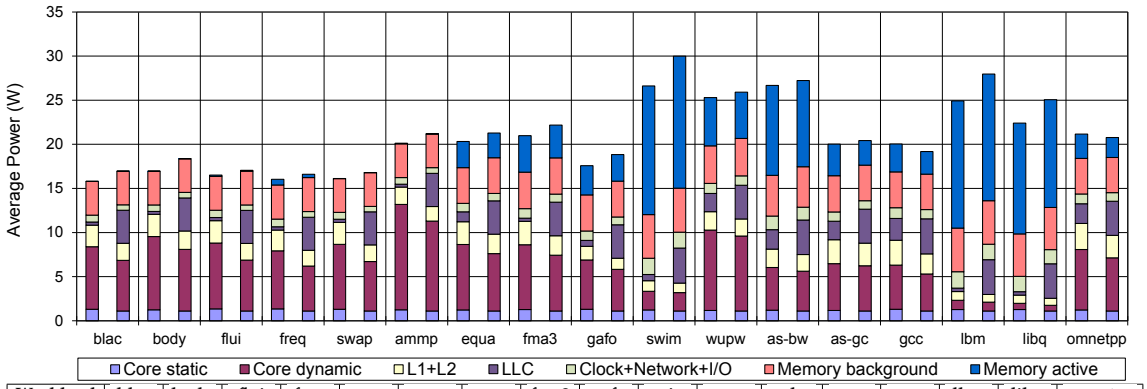
5.7.4 Results and Limitations

Equation 5.14 can be rearranged in the form $f(\gamma) \leqslant 0$ and solved in software using known techniques for solving cubic equations. To keep overheads low, we recommend using enumeration for $f(\gamma)$: for each allowed frequency step, evaluate $f(\gamma)$ and check the sign of the result. A change in sign between consecutive steps indicates a solution point. Each evaluation takes < 5 µsec on a Nehalem (2.26GHz) machine. We have 11 frequency steps over the frequency range that we consider. Thus, the DVFS computation time per cache configuration is < 0.06 msec and < 0.3 msec over all 5 cache configurations.

Figure 5.19 shows the performance gains and power-budget utilization of the best predicted configurations when optimizing performance subject to a system power budget. The baseline is the system with the highest-power LLC (32MB 32-way). The Table at the bottom of the figure includes the configurations selected.

Fifteen workloads show 0.5% to 15.2% performance improvement over the baseline. There is some under-utilization of the baseline power budget, leading to positive values of PwrSave numbers, due to conservative assumptions about the effect of DVFS on combinational logic and errors in expected performance and power of the target configurations. All 17 workloads for which a configuration different than the baseline was predicted saved energy (EnrSaveW) from 1.3% to 14.6%. The remaining workloads continued with the baseline configurations and did not save energy. The geometric mean of energy savings over all 32 workloads was 4.4%.

Unfortunately, the predicted configurations led to performance loss for swim and lbm and more power being used for gcc and omnetpp. Inaccurate predictions may lead to performance loss, under/over-utilization of the power budget and associated thermal overshoot. This situation is not unique to our system: any non-oracular predictive mechanism would need to detect and deal with occasional mispredictions.



Workload	blac	body	flui	freq	swap	ammp	equa	fma3	gafo	swim	wupw	as-bw	as-gc	gcc	lbm	libq	omnetpp
CacheSize	2MB	2MB	2MB	2MB	2MB	2MB	8MB	2MB	4MB	4MB	16MB	16MB	16MB	16MB	2MB	2MB	16MB
FreqScale	14.1%	9.4%	16.4%	16.4%	14.1%	9.4%	9.4%	11.7%	14.1%	9.4%	4.7%	7.0%	4.7%	14.1%	11.7%	11.7%	9.4%
PerfGain	14.1%	9.3%	15.2%	9.2%	14.0%	10.1%	7.4%	7.4%	8.1%	-3.3%	3.9%	2.3%	0.5%	7.6%	-1.1%	2.4%	6.5%
PwrSave	5.9%	6.8%	2.0%	2.4%	3.0%	4.1%	3.5%	4.2%	5.4%	10.8%	1.9%	1.6%	1.7%	-5.4%	10.4%	10.2%	-2.6%
EnrSave	17.5%	14.7%	14.9%	10.6%	14.9%	12.9%	10.2%	10.8%	12.5%	7.8%	5.6%	3.8%	2.2%	2.1%	9.4%	12.3%	3.6%
EnrSaveW	14.6%	11.4%	13.9%	9.3%	13.4%	11.1%	8.6%	9.0%	9.8%	3.6%	4.9%	3.2%	1.3%	4.7%	5.3%	7.9%	4.8%

Figure 5.19: System power budgeting (Section 5.7.3) with best predicted configuration (NI prediction). Two power stacks are shown for each workload: the left stack for the predicted configuration and the right stack for the baseline configuration (32MB 32-way LLC, 2132 MHz frequency). For each workload, the system power consumed by the baseline configuration is the power budget. **CacheSize** and **FreqScale** (= $\frac{\text{new frequency}}{\text{baseline frequency}}$ - 1) together define the predicted configuration. **PerfGain** shows performance gain (= speedup - 1), **PwrSave** shows system power saved (= remaining system power budget with respect to the baseline), and **EnrSave** shows system energy saved with the predicted configuration compared to the baseline. **EnrSaveW** accounts for wall power in energy savings calculations. Negative improvements (= losses) are highlighted.

We will now compare our model predictions to that by an oracle. The oracle runs all possible configurations (#configurations = #cache sizes $\times$ #frequencies) to determine the highest-performing configuration that satisfies the power budget. The oracle identifies configurations on the Pareto frontier. Figure 5.20 shows the performance gains with respect to the baseline configuration with the oracle and our model for all predictions where at least 2% gains were predicted, either by the oracle or the model, while being within the power budget. For example, the oracle predicts 16.5% gains for blac whereas

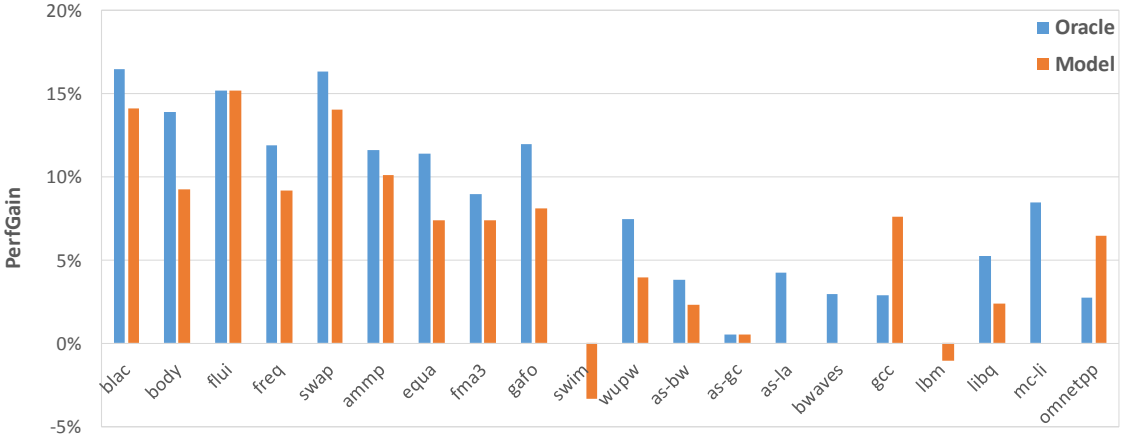


Figure 5.20: Comparison of performance gains between the oracle and our model.

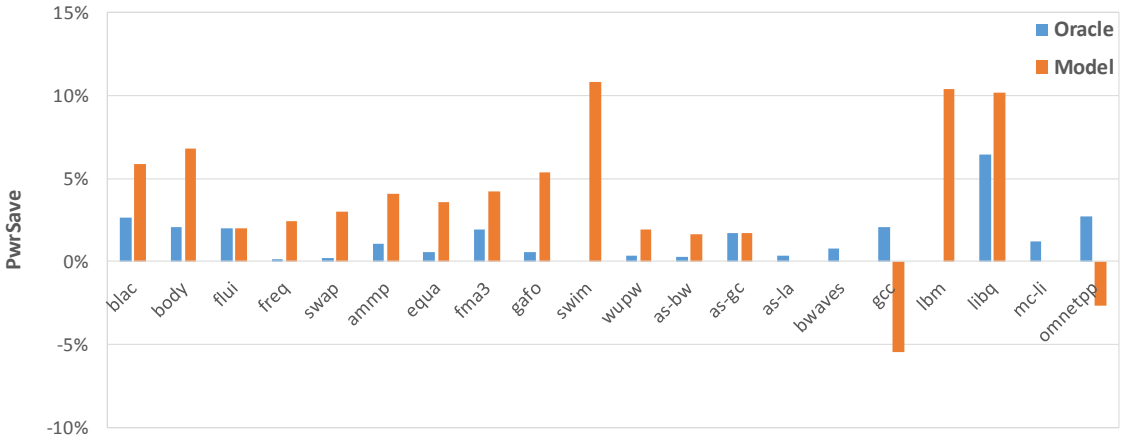


Figure 5.21: Comparison of power savings between the oracle and our model.

the configuration selected by our model achieved 14.1%. The oracle never selects a performance-losing configuration. Our model, on the other hand, selected configurations that caused performance losses on two applications—swim (3.3%) and lbm (1.1%). The model-selected configurations performed better than the oracle for two benchmarks—gcc

	Oracle PerfGain > Model PerfGain	**Oracle PerfGain < Model PerfGain**
Oracle PwrSave > Model PwrSave	as-la, bwaves, mcf-li	gcc, omnetpp
Oracle PwrSave < Model PwrSave	blac, body, freq, swap, ammp, equa, fma3, gafo, swim, wupw, as-bw, lbm, libq	

Figure 5.22: Comparison matrix between the oracle and our model.

(model: 7.6%, oracle: 2.9%) and omnetpp (model: 6.5%, oracle: 2.8%), because they overshot the power budget by 5.4% and 2.6% respectively. The power savings, shown in Figure 5.21, shows what percentage of the power budget remains unutilized. The savings are negative, indicate overshoot of the power budget, for gcc and omnetpp. For a number of workloads, the unutilized power budget is higher with the model-selected configurations than with the oracle-selected configurations. This under-utilization leads to lower performance gains than the oracle for the corresponding workloads in Figure 5.20.

Figure 5.22 shows a qualitative comparison between the oracle and the model, correlated across performance gains and power savings (power budget under-utilization). There cannot be any workload for which the oracle gains less performance and saves less power than the model, so the bin in the lower-right quadrant is empty. The bin in the upper-left quadrant shows workloads (as-la, bwaves, mcf-li)) where the oracle gained more performance as well as saved more power than the model. The upper-right quadrant shows workloads for which the model performed better than the oracle, but saved less power. This can only happen if the power budget is overshot. The lower-left shows workloads where the model resulted in conservative behavior—less performance than the oracle, but more underutilized power budget. The remaining workloads had identical gains with the oracle and with the model. This includes two workloads, flui and as-gc,

with performance gains (15.2% and 0.5% respectively) with power budgeting and the rest of the workloads with no gains or less than 2% gains. Note that since gains for as-gc are within the minimum 2% threshold, this configuration is not actually selected by the oracle, but we include it here to maintain the invariant that the oracle is always at least as good as the model.

Of the workloads show in Figures 5.20 and 5.21, 11 workloads (blac, body, flui, swap, ammp, fma3, as-bw, as-gc, gcc, libq, omnetpp) had the same cache size selected by both the model and the oracle. Of these, flui and as-gc also had the same frequency selections. For gcc and omnetpp, the frequency selected by the model was higher than that selected by the oracle resulting in overshoot of the power budget. For the remaining 7 workloads, the model-selected frequency was lower than the oracle-selected frequency resulting in under-utilization of the power budget. Since frequency reconfigurations can be done with significantly less overhead than cache reconfigurations, they can be subsequently adjusted to stay within or better utilize the power budget.

We propose using online monitoring schemes similar to Intel's Turbo Boost technology [111] so that the system can continuously monitor and compare predicted power-performance with actual values. In case performance is below expectations, the system will revert back to baseline thereby restoring long-term performance loss to 0%. In case of power/thermal overshoot with strict budgets, throttling mechanisms must be employed to scale down excess voltage and frequency immediately. For soft budgets, corrective action can be taken in the next interval. A guard mechanism may be used along with our predictors to make the system energy-secure [36].

Our DVFS model assumes that enough thermal headroom is available to accommodate the desired scaling. We claim that the maximum permissible scaling is limited by how efficiently heat can be moved away from each individual core to the heat sink, and is

not significantly lowered by our scheme. Thermal resistance is directly proportional to thickness/separation and inversely proportional to cross-sectional area [199]. Typical chip thicknesses are < 1 mm and thermal conductivity of copper is higher than silicon [199]. Thus, lateral thermal resistance between cores is expected to be much higher than vertical thermal resistance. As long as on-chip/system TDP is not exceeded, which is guaranteed when predictions are accurate, any system that supports overscaling of voltage/frequency of cores should be able to benefit from LLC power budgeting for performance. Although we assumed a maximum frequency scaling of 25%, the maximum actual scaling was 16.4%. A lower scaling limit would reduce speedups and is similar to having a lower power margin.

Limitations of the reuse-based cache performance estimation framework, discussed in Section 4.8 of Chapter 4, continue to be applicable to this study.

5.8 Conclusion

As technology scales, intelligently budgeting power between system components will become increasingly important to obtaining optimum power-performance. In this chapter, we focused on maximizing performance of a chip multiprocessor (CMP) system for a given power budget (SLApower), by developing techniques to budget power between processor cores and caches. While this governor only targeted SLApower, it can also be easily retargeted for SLAee or SLAperf.

Dynamic cache configuration can reduce cache capacity, thereby freeing up chip power, but may increase the miss rate (and potentially memory power). Dynamic voltage and frequency scaling (DVFS) can exploit the saved power to increase core performance, potentially increasing system performance. We demonstrated that favorable configurations can be selected using simple analytic models, driven by hardware performance counters

to estimate the cache reuse distribution (also see Chapter 4).

Unavailability of this reconfiguration knob in real systems made it necessary to evaluate this space using full-system simulation. Detailed simulation models show that carefully budgeting power between cores, memory, and caches can improve system performance 0.5%–15.2% for 15 of 32 workloads and save an average of 4.4% total energy (wall power) averaged over all 32 workloads.